ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ АКАДЕМИЧЕСКИЙ УНИВЕРСИТЕТ РОССИЙСКОЙ АКАДЕМИИ НАУК

На правах рукописи

Диссертация допущена к защите
Зав. кафедрой
_____
“ ”_____20... г.

Д И С С Е Р Т А Ц И Я
НА СОИСКАНИЕ УЧЕНОЙ СТЕПЕНИ
М А Г И С Т Р А

Тема: Работа с файловыми системами в операционной системе Windows с использованием драйверов операционной системы Linux

Направление: 03.04.01 – Прикладные математика и физика

Выполнил студент *Новокрещенов К.С.*
(подпись)

Руководитель:
магистр прикладной математики и физики *Баталов Е.А.*
(подпись)

Рецензент:
специалист *Борзенков П.А.*
(подпись)

Санкт-Петербург
2015 г.

Содержание

1 Введение

1.1 Файловые системы. Определение, функции, виды

Файловая система является ключевым компонентом операционной системы и зачастую играет решающее значение при выборе той или иной операционной системы. Файловую систему можно рассматривать как способ организации, хранения и именования данных на носителях информации в компьютере или любом другом электронном оборудовании. Без наличия файловой системы, вся информация размещалась бы на устройстве хранения данных произвольным образом, в виде одного непрерывного блока, без возможности определения, в каком месте заканчивается один блок информации и начинается другой. Файловая система определяет формат содержимого и способ физического хранения информации, которую принято группировать в виде файлов (логически неделимой совокупности данных). Таким образом, файловая система осуществляет контроль за тем, где и как файлы сохраняются на физическом устройстве и по каким правилам извлекаются обратно для использования в работе приложений.

Файловая система не только реализует определенный набор логических правил, по которым файлы размещаются на устройстве, но также хранит дополнительную информацию о каждом файле, используемую для его однозначной идентификации (имя файла) и получения сведений о некоторых его свойствах (размер, время создания, время последней модификации и т.д.). Файловые системы управляют доступом к файлам и их метаданным.

Файловые системы могут использоваться на различных устройств. Каждое устройство хранения данных использует определенный тип физического носителя. Среди них можно выделить носители с произвольным доступом такие как жесткий диск, носители с последовательным доступом (магнитные ленты), оптические носители, флэш-память и другие. Наиболее распространенными на сегодняшний день являются жесткие диски. Файловая система связывает носитель информации с одной стороны и программный интерфейс для доступа к файлам - с другой.

Файловая система не обязательно напрямую связана с физическим носителем информации. Существуют виртуальные файловые системы (например, procfs), файлы и каталоги внутри которых генерируются при запросе, а также сетевые файловые системы (например, Network File System (NFS)), которые являются лишь способом доступа к файлам, находящимся на удалённом компьютере.

Некоторые файловые системы были разработаны специально для решения определенного класса задач и использовании в работе конкретных приложений. Так, например, файловая система ISO 9660 была спроектирована исключительно для оптических дисков, а файловая система VMFS представляет собой кластерную файловую систему, предназначенную для хранения других файловых систем.

На сегодняшний день IT-индустрия «перегружена» огромным количеством разнообразных файловых систем. Каждая файловая система имеет свою определенную логическую структуру, характеризуется скоростью работы, надежностью и способностью к восстановлению после возникновения критических ошибок, сбоев и отказов оборудования, предоставляемой безопасностью одновременного доступа к файлам, занимаемым размером на физическом устройстве, возможностью гибкой настройки в зависимости от используемого физического носителя данных и т.д.

Каждая файловая система имеет свои преимущества и недостатки по сравнению с другими файловыми системами. Выбор той или иной файловой системе зависит от тех задач, в решении которых она будет использоваться. Например, в случае почтового сервера или баз данных хорошо подойдет файловая система, спроектированная специально для работы с файлами малого размера, а при использовании в маломощных компьютерных установках отличным выбором будет использование файловой системы, потребляющей малое количество процессорного времени и не оказывающей большой нагрузки на процессор.

1.2 Нативные файловые системы

Для каждой операционной системы существует набор файловых систем, поддержка которых изначально включена в ядро операционной системы. Такие файловые системы являются нативными для данной операционной системы. Как правило, изначально каждая новая файловая система разрабатывается специально для использования в конкретной операционной системы. Наиболее известными примерами таких файловых систем являются файловые системы семейства Ext4 для операционной системы Linux, файловая система NTFS операционной системы Windows, файловая система HFS операционной системы Mac OS и многие другие.

Количество нативно поддерживаемых файловых систем сильно варьируется в зависимости от рассматриваемой операционной системы. Так, например, основное ядро операционной системы Linux поддерживает широкий спектр разнообразных файловых систем, в то время как в операционной системе Windows включена поддержка сравнительно небольшого количества файловых систем. Рассмотрим более подробно какие файловые системы поддерживаются операционными системами Windows и Linux.

1.2.1 Файловые системы операционной системы Linux

За всё время существования и эволюции операционной системы Linux в её основное ядро была добавлена поддержка огромного количества разнообразных файловых систем[1]. Рассмотрим наиболее популярные и известные из них[2].

[1] http://en.wikipedia.org/wiki/Category:Linux_kernel-supported_file_systems
[2] http://www.howtogeek.com/howto/33552/htg-explains-which-linux-file-system-should-you-choose

1.2.1.1 Семейство файловых систем Ext

Файловая система Ext

Первая файловая система, разработанная специально для нужд операционной системы Linux. Была создана в 1992 с целью преодолеть ограничения существовавшей в то время файловой системы Minix. На сегодняшний день файловая система Ext является устаревшей и уже не поддерживается во многих дистрибутивах Linux.

Файловая система Ext2

Нежурналируемая файловая система, представляет собой продолжение развития файловой системы Ext2. Включена поддержка расширенных файловых атрибутов, увеличен максимальный поддерживаемый размер физического устройства (до 2 терабайт). Поскольку журналирование не используется, в процессе своей работы данная файловая система осуществляет существенно меньше операций записи на диск, вследствие чего хорошо подходит для использования в устройствах с флеш-памятью.

Файловая система Ext3

Дальнейшее развитие семейства Ext* файловых систем привело к появлению журналируемой файловой системы Ext3. По сути является расширением файловой системы Ext2, способное к журналированию. Одной из главных целей при разработке данной файловой системы была поддержка обратной совместимости. Вследствие этого переход с файловой системы Ext2 к файловой системе Ext3 не требует форматирования. На сегодняшний день Ext3 является наиболее стабильной и поддерживаемой файловой системой в среде Linux.

Файловая система Ext4

Представляет собой своеобразную попытку создать 64-х битную Ext3, способную поддерживать больший размер файловой системы. Также добавились такие возможности как непрерывные области дискового пространства, задержка выделения пространства, онлайн дефрагментация и прочие. Обеспечивается прямая совместимость с системой Ext3 и ограниченная обратная совместимость при недоступной способности к непрерывным областям дискового пространства.

1.2.1.2 Файловая система ReiserFS

Первая попытка создать файловую систему нового поколения для Linux. Данная файловая система, представленная в 2001 году, включает в себя журналирование, возможность динамического масштабирования и лишена многих недостатков, присутствующих в файловых системах семейства Ext. Данная файловая система показывает прекрасную производительность при работе с маленькими файлами и логами, поэтому идеально подходит для использования в базах данных и почтовых серверах.

1.2.1.3 Файловая система BtrFS

Файловая система, изначально разрабатываемая компанией Oracle, основана на структурах B-деревьев и работает по принципу копирование-при-записи. Предоставляет такие возможности как снимки файловой системы, контроль за целостностью данных и метаданных, прозрачное сжатие данных, журналирование, поддержка оптимизированного режима при использовании в SSD-накопителях, дефрагментация в рабочем режиме и многое другое. Немаловажным фактом является возможность перехода с файловых систем Ext3 и Ext4 на BtrFS без необходимости форматирования. Данная файловая система лишена многих недостатков, присущим предшествующим файловым системам операционной системы Linux, и призвана стать файловой системой Linux по умолчанию.

1.2.1.4 Файловая система XFS

Файловая система, разработанная компанией Silicon Graphics в 1994 году и портированная на Linux в 2001 году. Во многом похожа на файловую систему Ext4, реализует журналирование, задержку выделения пространства (как метод борьбы с фрагментацией), онлайн дефрагментацию, обладает возможностью динамического расширения. Показывает хорошую производительность при работе с большими файлами, но в остальном не обладает существенными преимуществами по сравнению с Ext4.

1.2.1.5 Файловая система JFS

Файловая система, разработанная компанией IBM в 1990 году и позже портированная на Linux. Главными особенностями данной файловой системы является низкий уровень потребления процессорного времени и хорошая производительность при работе с файлами как большого, так и малого размера. Как и файловые системы XFS и ReiserFS, предоставляет возможность динамического масштабирования, поддерживает журналирование. Вследствие низкого уровня использования процессора, хорошо подходит для применения в работе маломощных компьютеров и серверов.

1.2.1.6 Файловая система ZFS

Файловая система, разработанная компанией Sun Microsystems, первоначально для операционной системы Solaris. Среди отличительных особенностей можно выделить отсутствие фрагментации данных как таковой, возможность по управлению снимками файловой системы, пулами хранения. ZFS показывает прекрасные результаты производительности при работе с большими дисковыми массивами. Поскольку данная файловая система является проприетарной, она не может быть включена в основное ядро операционной системы Linux. Тем не менее, в Linux-системах ZFS может использоваться с помощью фреймворка FUSE[1].

[1] http://fuse.sourceforge.net/

1.2.2 Файловые системы операционной системы Windows

По сравнению с операционной системой Linux, операционная система Windows поддерживает существенно меньшее количество файловых систем[1]. К ним относятся файловые системы FAT[2], NTFS[3], exFAT[4].

1.2.2.1 FAT

Классическая архитектура файловой системы, которая из-за своей простоты всё ещё широко используется для флеш-накопителей и карт памяти. Данное семейство файловых систем (8-bit FAT, FAT-12, FAT-16, FAT-32) поддерживается почти всеми операционными системами для персональных компьютеров, включая все версии операционной системы Windows и MS-DOS. Является в некотором смысле универсальной файловой системой, предназначенной для обмена данными между компьютерами и устройствами практически любого типа.

1.2.2.2 NTFS

NTFS заменила ранее использовавшуюся в MS-DOS и Windows файловую систему FAT. Впервые появилась в операционной системе Windows-NT, выпущенной в 1993 году. NTFS поддерживает систему метаданных и использует специализированные структуры данных для хранения информации о файлах для улучшения производительности, надёжности и эффективности использования дискового пространства. NTFS имеет встроенные возможности разграничения доступа к данным для различных пользователей и групп пользователей (списки контроля доступа – Access Control Lists, ACL), а также позволяет назначать квоты (ограничения на максимальный объём дискового пространства, занимаемый теми или иными пользователями). NTFS использует систему журналирования для повышения надёжности файловой системы. Также среди особенностей данной файловой системы можно выделить поддержку жестких ссылок, разреженных файлов, шифрования, сжатия данных, точке повторной обработки (специальный тип каталогов, используемых как точки монтирования других файловых систем, символических ссылок, ссылок удаленных файловых систем).

1.2.2.3 exFAT

Проприетарная файловая система, является преемником файловой системы FAT-32. Разработана компанией Microsoft преимущественно для мобильных носителей таких как флеш-память, SSD-диски, смарткарты. Позволяет хранить файлы и использовать разделы значительно большего размера, чем в файловых системах семейства FAT. В системе exFAT также появилась возможность управления правами доступа на файлы/каталоги, а время

[1] http://en.wikipedia.org/wiki/File_system#Microsoft_Windows
[2] https://ru.wikipedia.org/wiki/FAT
[3] https://ru.wikipedia.org/wiki/NTFS
[4] http://en.wikipedia.org/wiki/ExFAT

доступа к данным уменьшилось. Файловую систему exFAT можно считать конкурентом NTFS на системах с ограниченной вычислительной мощности и памяти. Поддерживается в Windows XP, Windows Vista, Windows 7, Windows 8.

1.3 Необходимость поддержки ненативных файловых систем

Необходимость работы с файловыми системами, не являющимися нативными для используемой операционной системы, может возникнуть в ряде случаев.

Зачастую в рамках решения той или иной задачи наиболее выгодно использовать конкретную файловую систему из-за предоставляемый ею преимуществ (высокая скорость работы, поддержка журналирования, возможность шифрования данных и т.д.) по сравнению со другими файловыми системами, даже если выбранная файловая система не является нативной в используемой операционной системе. При этом не всегда удобно отказаться от используемой операционной системы в пользу той, в которой требуемая файловая система поддерживается в качестве нативной.

Многим прикладным программам, работающим в той или иной операционной системе, не всегда достаточно поддерживаемого операционной системой набора файловых систем. Существуют целые классы программного обеспечения, которым в силу своей специфики требуется работать с широким количеством файловых систем. К ним относятся:

- Системы резервного копирования и восстановления данных
- Системы защиты данных, антивирусное программное обеспечение
- Менеджеры жестких дисков и разделов
- Файловые менеджеры – программы для просмотра и управления содержимым файловой системы, реализующие возможность копирования, перемещения, удаления файлов и каталогов внутри файловой системы

В случае если поддержка файловый системы не реализована на уровне операционной системы, задача организации доступа к ней должна решаться в рамках разрабатываемого программного обеспечения.

При этом важно помнить, что чем с большим количеством ненативных файловых систем способно работать приложение, тем оно более конкурентоспособно и привлекательно для пользователя.

2 Постановка и анализ задачи

2.1 Цель работы

На сегодняшний день, операционные системы Windows и Linux являются одними из самых распространенных. Следовательно, это становится верным и для файловых систем, которые они используют в своей работе. Важно заметить, что множество нативных файловых систем ОС Windows является подмножеством файловых систем ОС Linux. Поэтому в случае необходимости приложению, работающему в операционной системе Linux, получить/обеспечить доступ к содержимому файловых систем, нативных для операционной системы Windows, не нужно прилагать дополнительных усилий. Практически все файловые системы ОС Windows поддерживаются основным ядром ОС Linux.

Однако при необходимости приложению Windows получить доступ к нативным файловым системам операционной системы Linux всё становится гораздо сложнее. Практически все файловые системы, нативные для ОС Linux, не поддерживаются в ОС Windows. В этом случае проблема организации доступа к требуемой файловой системе ОС Linux должна решаться самим Windows-приложением.

В настоящей работе решается проблема организации доступа к нативным файловым системам ОС Linux внутри ОС Windows.

Цель настоящей работы заключается в предоставлении приложениям операционной системы Windows возможности работать с файловыми системами, нативными для операционной системы Linux, но не поддерживаемыми операционной системой Windows.

2.2 Организация доступа к ненативным файловым системам

Для организации работы с любой файловой системы необходим так называемый «драйвер» файловой системы. Драйвер файловой системы можно охарактеризовать как некоторый программный компонент, интерпретирующий структуры файловой системы и предоставляющий использующим её приложениям логический иерархический вид её содержимого. Драйвер может являться частью операционной системы (в случае нативных файловых систем) либо поставляться сторонним производителем как отдельный программный модуль.

Для возможности работы с нативной файловой системой отдельному приложению не нужно прилагать никаких усилий. Как уже упоминалось выше, операционная система изначально имеет встроенные в её ядро драйвера, организующие полный доступ к нативным файловым системам, включающий в себя возможность чтения, записи, модификации содержащихся в файловой системе каталогов и файлов. Стоит отметить, что

нативные драйвера файловых систем, как правило, пишутся, тестируются и поддерживаеются самими разработчиками операционной системы, в которой данная файловая система является нативной. Вследствие этого, такие драйвера обладают высокой надежностью, производительности, эффективно реализуют все возможности файловой системы, своевременно обновляются и активно поддерживаются разработчиками в течение всего времени существования файловой системы как нативной.

В случае необходимости работы с ненативной файловой системой приложение должно предоставить операционной системе драйвер этой файловой системы. Задача поиска и реализации такого драйвера должна решаться разработчиками данного приложения.

Как и в случае нативных файловых систем, для предоставления приложению возможности работать с файловой системой, не являющейся нативной, необходимо использовать драйвер этой файловой системы. Таким образом, задача поддержки ненативной файловой системы ложится на плечи разработчиков приложения (программы).

Можно выделить два принципиально отличающихся друг от друга подхода к решению данной проблемы:

1. Использование «нативного» драйвера файловой системы, то есть драйвера, реализованного для той операционной системы, в которой данная файловая система является нативной
2. Использование «портированного» драйвера файловой системы, то есть драйвера, реализованного специально для работы в целевой операционной системе.

Поскольку в данной работе исследуется возможность поддержки нативных файловых систем Linux в операционной системе Windows, рассмотрим оба подхода более детально на примере операционных систем Windows и Linux. В данном случае операционная система Linux является исходной (нативной), и драйвера, используемые в ней для доступа к её нативным файловым системам, мы будем называть «нативными». При этом операционная система Windows будет выступать в качестве целевой, соответственно, реализованные для работы в ней драйвера, обеспечивающие доступ к нативным файловым системам Linux, мы будем называть «портированными». Также особое внимание будет сосредоточено на анализе уже существующих в рамках каждого подхода примеров решений.

2.2.1 Использование портированного драйвера файловой системы

В рамках данного подхода существуют несколько возможных решений. Рассмотрим каждое из них в отдельности и приведем существующие примеры каждого из решений при наличии таковых.

2.2.1.1 Использование драйвера файловой системы, реализованного «с нуля»

Отличительной особенностью подавляющего числа современных файловых систем является их высокая сложность, непростое внутреннее устройство, использование в своей работе трудных и сложных алгоритмов. Это объясняется тем, что к вновь появляющимся файловым системам предъявляются высокие требования по скорости доступа к данным, эффективному использованию ресурсов, предоставляемой безопасности и многие другие. Каждая новая файловая система должна быть производительнее своих предшественников, более эффективно решать поставленные перед ней задачи, чтобы быть конкурентоспособной и более привлекательной для конечного пользователя.

Сложность внутренней организации современных файловых системы и используемых ими алгоритмов работы оказывают прямое влияние на сложность реализации полноценных драйверов файловых систем. Разработка драйвера современной файловой системы, например, такой как BtrFS, представляет собой достаточно трудоемкий процесс, требующий больших затрат ресурсов и времени. Для реализации надежного и эффективного драйвера требуется не только глубокое изучение внутреннего устройства файловой системы и принципов её работы, но и всестороннее знание особенностей операционной системы, в которой он будет работать. Стоит также упомянуть о необходимости длительного тестирования правильной работоспособности полученного драйвера перед его использованием с целью выявления допущенных при его разработке ошибок и недоработок. Также необходимо постоянно следить за изменениями файловой системы и своевременно обновлять созданную версию драйвера.

Таким образом, одним из главных недостатков данного решения является его высокая сложность и трудоемкость, обремененная необходимостью постоянно поддерживать реализованный драйвер в актуальном состоянии при обновлениях файловой системы. В случае необходимости одновременной поддержки нескольких файловых систем затраты на реализацию данного решения возрастают пропорционально количеству поддерживаемых файловых систем. К преимуществам данного решения можно отнести потенциально высокую производительность работы полученного драйвера, полностью реализующего всю функциональность файловой системы.

2.2.1.2 Использование драйвера файловой системы, реализованного сторонними разработчиками

В данном случае в зависимости от «происхождения» драйвера возможны два варианта: использование свободно распространяемого драйвера либо использование драйвера, реализованного коммерческой организацией.

В большинстве случаев, качество и надежность свободно распространяемых драйверов оставляет желать лучшего. Как правило, разработка таких драйверов осуществляется непрофессиональными программистами и чаще всего носит чисто экспериментальный характер. Зачастую в таких драйверах реализуются лишь некоторые базовые функции

файловой системы, многое остается недоделанным и недоработанным. Также характерной особенностью таких драйверов является отсутствие тщательного тестирования и поддержки со стороны разработчиков. Ошибки, допущенные при разработке такого драйвера, в процессе его использования могут привести к самым нежелательным последствиям: от ошибки времени выполнения и нежелательной перезагрузки операционной системы до отказа и повреждения оборудования с частичной или полной потерей данных. В любом случае, задачу доработки, тестирования и своевременного обновления драйвера необходимо решать конечному пользователю данного драйвера. На данный момент для некоторых файловых систем операционной системы Linux крайне сложно найти хорошую реализацию полноценного драйвера, предназначенного для работы в операционной системе Windows, иногда такого драйвера просто не существует.

Коммерческие драйвера файловой системы, как правило, лишены недостатков, связанных со свободно распространяемыми драйверами. Главным же их недостатком, очевидно, является необходимость материальных затрат на покупку лицензии для использования драйвера и поддержки со стороны разработчиков. В случае необходимости полноценного доступа к большому набору файловых систем данное решение может оказаться слишком невыгодным.

2.2.1.3 Использование отдельных приложений и утилит, реализующих частичный или полный доступ к файловой системе

Существуют отдельные приложения операционной системы Windows, реализующие доступ к некоторым файловым системам, не являющимися нативными. Чаще всего такие приложения реализуют ограниченную функциональность и выступают в качестве файловых обозревателей, позволяя лишь просматривать содержимое файловой системы, осуществлять чтение и копирование файлов. Главным недостатком при использовании таких программ является невозможность работать с файловой системой из других приложений. Для полноценной работы с содержащимися в файловой системы файлами и каталогами требуется вручную скопировать интересующие файлы на поддерживаемую операционной системой файловую систему и использовать в последующей работе только их копии. В качестве преимущества данного способа можно указать на тот факт, что поскольку доступ к файловой системе выполняется в режиме чтения, гарантируется, что содержащиеся данные внутри файловой системы не будет повреждены, и сама файловая система останется в неизменном работоспособном состоянии.

2.2.1.4 Примеры существующих решений

Подавляющее большинство из существующих на данный момент драйверов операционной системы Windows, предназначенных для работы с файловыми системами операционной системы Linux, реализовано для доступа к файловым системам семейства Ext[1].

[1] http://www.howtogeek.com/112888/3-ways-to-access-your-linux-partitions-from-windows

Драйвер Ext2fsd[1]

Свободно распространяемый драйвер операционной системы Windows для доступа к Ext2, Ext3 и Ext4 файловым системам. Предназначен для использования в Windows XP, Windows Vista, Windows 7. Предоставляет возможность работать с перечисленными файловыми системами нативно, обеспечивая другим приложениям доступ к файловой системе через букву диска[2].

Основные особенности драйвера Ext2fsd:

- поддержка чтения и записи Ext2 и Ext3 файловых систем
- поддержка чтения Ext4 файловой системы
- отсутствие полной поддержки журналирования файловой системы Ext3
- отсутствие поддержки менеджера логических томов операционной системы Linux
- отсутствие поддержки шифрования

Данный драйвер является наиболее распространённым среди драйверов такого типа, в большинстве случаев работает стабильно, однако иногда при записи возможны потери данных[3].

Драйвер Ext2 Installable File System[4]

Свободно распространяемый драйвер операционной системы Windows для работы с файловыми системами семейства Ext. Предназначен для использования в следующих операционных системах семейства Windows: Windows NT, Windows XP, Windows Vista. Представляет собой драйвер файловой системы Ext2, полностью поддерживающий операции чтения и записи. Работает в режима ядра, то есть на том же программном уровне, что и нативные драйвера операционной системы Windows. Драйвер позволяет назначить разделам с файловой системой Ext2 букву диска, что предоставляет возможность всем приложениям взаимодействовать и работать с ними. Поскольку файловая система Ext3 поддерживает обратную совместимость с файловой системой Ext2, данный драйвер также может использоваться для организации работы с Ext3, но без поддержки журналирования. К основными недостатками данного драйвера можно отнести:

- отсутствие поддержки журналирования в файловой системе Ext3
- отсутствие поддержки прав доступа
- отсутствие возможности производить дефрагментацию файловой системы, как и возможности получения информации о текущей фрагментации раздела
- отсутствие поддержки менеджера логических дисков операционной системы Linux

[1] http://www.ext2fsd.com/
[2] http://en.wikipedia.org/wiki/Drive_letter_assignment
[3] http://sourceforge.net/projects/ext2fsd/reviews
[4] http://www.fs-driver.org/

Драйвер Paragon ExtFS for Windows[1]

Программное обеспечение компании Paragon, предоставляющий полноценный доступ к файловым системам Ext2, Ext3 и Ext4. Ключевые особенности:

- обеспечивает быстрый и прозрачный доступ к файловым системам семейства Ext*, полностью поддерживает операции чтения и записи
- предоставляет инструменты для создания и форматирования разделов с файловым системами семейства Ext*
- поддерживает работу с LVM разделами только в режиме чтения
- поддерживает последнюю версию операционной системы Windows 8.1
- поскольку работает в режиме ядра, обеспечивает хорошую производительность и скорость работы

Приложение DiskInternals Linux Reader[2]

Приложение с графическим пользовательским интерфейсом операционной системы Windows для просмотра содержимого нативных файловых систем операционной системы Linux. Поддерживает чтение таких файловых систем как Ext2, Ext3, RaizerFS, Reizer4, а также HFS+ операционной системы Apple OS. Не предоставляет остальным программам возможность работать с данными файловыми системами через символ диска. Поскольку поддерживается только чтение, гарантируется, что при использовании данной программы файловая система останется в неизменном состоянии и не будет повреждена.

Приложение Ext2Read[3]

Приложение операционной системы Windows с графическим интерфейсом похожим на интерфейс проводника Windows Explorer. Используется для просмотра и чтения содержимого Ext2, Ext3 и Ext4 файловых систем. Также поддерживает возможность чтения разделов под управлением менеджера логических томов Linux.

Приложение Explore2fs[4]

Графическое приложения для просмотра содержимого Ext2 и Ext3 файловых систем. Поддерживает только возможность чтения и копирования файлов.

Утилита LTOOLS[5]

Утилита командной строки, предназначенная для чтения и записи Ext2 и Ext3 файловых систем. Также поддерживает возможность работы с файловой системой ReiserFS.

[1] https://www.paragon-software.com/home/extfs-windows-pro
[2] http://www.diskinternals.com/linux-reader/
[3] http://sourceforge.net/projects/ext2read/
[4] http://sourceforge.net/projects/explore2fspe/
[5] http://www.it.hs-esslingen.de/~zimmerma/software/ltools/ltools.html

Проект ZFS-WIN[1]

Свободно распространяемый драйвер операционной системы Windows, предназначенный для доступа к файловой системе ZFS. Позволяет монтировать раздел с файловой системой ZFS только в режиме чтения и осуществлять доступ к нему из других приложений через присвоенную разделу букву диска.

Проект WinBtrfs[2]

Драйвер операционной системы Windows, работающий в пользовательском режиме, предназначен для доступа к файловой системе BtrFS. Позволяет приложениям операционной системы Windows работать с файловой системой BtrFS только в режиме чтения.

2.2.2 Использование нативных драйверов

Нативные драйвера файловой системы лишены многих недостатков, присущих их портированным на другие операционные системы аналогам. Такие драйвера полностью поддерживают все функции и особенности файловой системы, показывают более высокие результаты производительности, поскольку изначально проектируются для использования в конкретной операционной системе. Они более надежны, качественны и активно поддерживаются разработчиками как файловой, так и операционной системы, в которой эта файловая система является нативной. Используя нативные драйвера файловой системы, практически полностью пропадает необходимость дополнительного их тестирования и отслеживания изменений файловой системы. В случае с драйверами нативных файловых систем операционной системы Linux, прежде чем включить поддержку файловой системы в основное ядро Linux, драйвер этой файловой системы проходит тщательное тестирование огромным Linux-сообществом, в состав которого входят ведущие компании IT-индустрии.

Однако поскольку драйвер относится к типу программного обеспечения, сильно зависящего от операционной системы, его нельзя просто взять и начать использовать в другой, ненативной для него, операционной системе. Для реализации этой возможности необходимо в целевой операционной системе создать окружение, эмулирующее окружение нативной для данного драйвера операционной системы. Соответственно, для использования нативных драйверов файловых систем операционной системы Linux внутри операционной системы Windows требуется создать некоторое окружение ядра Linux, которое ожидает увидеть драйвер,

Таким образом, с одной стороны, проблема поиска надежного и эффективного драйвера оказывается полностью решенной, но, с другой стороны, возникает не менее тривиальная задача – эмуляции окружения одной операционной системы внутри другой.

[1] https://code.google.com/p/zfs-win
[2] https://github.com/jgottula/WinBtrfs

Данную проблему можно решить следующими способами:

- реализовать программный интерфейс (API) ядра нативной для драйвера операционной системы (в нашем случае Linux) в пространстве пользователя целевой операционной системы (в нашем случае Windows)
- использовать виртуальную машину

Очевидно, что сложность реализации первого способа колоссальна, особенно в случае необходимости эмуляции такой операционной системы как Linux. Разработка современной операционной системы представляет собой одну из самых сложных задач в IT-индустрии, а потому и портирование операционной системы для запуска в качестве отдельного приложения внутри другой во многом превосходит сложность портирования драйвера файловой системы.

Эмуляция окружение ядра Linux для работы нативного драйвера файловой системы операционной системы Linux в пользовательском пространстве Windows предполагает реализации как минимум:

- блочного уровня ядра Linux
- фреймворка файловых систем
- примитивов синхронизации
- фреймворка файловых систем в пользовательском пространстве (FUSE)

Также стоит отметить, что вследствие того, что API ядра Linux постоянно меняется, и эти изменения не всегда носят систематический и последовательный характер, необходим прилагать дополнительные усилия на поддержание эмулированного ядра в актуальном состоянии.

Использование виртуальной машины для эмуляции окружения одной операционной системы внутри другой позволит использовать нативную для драйвера операционную систему без изменений. В данном случае полностью исчезает проблема портирования как драйвера, так и нативной для него операционной системы. Главным недостатком такого подхода является тот факт, что использование виртуальной машины предполагает дополнительные издержки ресурсов системы (памяти и процессорного времени).

Несмотря на то, что при реализации первого подхода мы потенциально может получить менее ресурсоемкое и более производительное решение, использование виртуальной машины выглядит наиболее привлекательным с точки зрения потраченных усилий и получаемых выгод.

Именно такой подход был выбран для реализации возможности работы с нативными файловыми системами операционной системы Linux в операционной системе Windows.

2.3 Выбор решения

Для решения задачи организации доступа к нативным файловым системам операционной системы Linux в операционной системе Windows был выбран подход, заключающийся в использовании нативных драйверов Linux с эмуляцией окружения ядра Linux посредством использования виртуальной машины. Кратко перечислим основные преимущества данного подхода по сравнению с остальными (рассмотренными ранее):

- отсутствие затрат на разработку драйвера для каждой из файловой систем, доступ к которым необходимо организовать
- сложность реализации данного подхода никак не зависит от количества файловых систем, возможность работы с которыми нужно предоставить в операционной системе Windows
- нативные драйвера операционной системы Linux реализуют полный доступ к файловой системе, поддерживают все её функции. Обладают высокой производительностью и надёжностью, активно развиваются и поддерживаются разработчиками, лишены ошибок и недоработок, зачастую встречающихся в их портированных аналогах, реализованных сторонними компаниями и отдельными программистами
- драйвера операционной системы Linux, как и сама операционная система Linux, относятся к свободно распространяемому программному обеспечению, следовательно, отсутствуют материальные затраты, связанные с их использованием
- полное переиспользование исходного кода операционной системы Linux, отсутствие затрат на портирование

Одним из главных недостатков данного подхода является существенные потери в производительности, а именно скорости выполнения операций чтения и записи. Также происходит высокое потребление системных ресурсов, связанных с тем, что осуществляется запуск целой виртуальной машины, внутри которой работает полноценная операционная система Linux. Существует несколько способов устранения данных недостатков, о них будет рассказано далее более подробно

Реализация выбранного подхода будет выполнена в виде динамической библиотеки операционной системы Windows, написанной на языке программирования C. Выбор языка C в качестве языка разработки является наиболее целесообразным, поскольку:

- язык программирования C является одним из наиболее распространенных
- код, написанный на языке C, потенциально обладает более высокой производительностью в связи с отсутствием дополнительных накладных расходов связанных с использованием интерпретатора, сборщика мусора и т.д., присутствующих в других языках программирования высокого уровня

- многие языки программирования высокого уровня предоставляют возможность использовать библиотеки, написанные на C, что расширяет сферу применения реализуемой библиотеки

Разрабатываемая библиотека должна предоставлять приложениям операционной системы Windows определенный программный интерфейс для возможности осуществления таких базовых операций как:

1. Чтение нативных файловых систем операционной системы Linux, расположенных как на реальном физическом устройстве, так и внутри образа диска. Под «чтением файловой системы» подразумевается просмотр содержимого каталогов, чтение и копирование файлов.
2. Запись нативных файловых систем операционной системы Linux, расположенных как на реальном физическом устройстве, так и внутри образа диска. Под «записью файловой системы» подразумевается изменение содержимого каталогов, редактирование файлов, создание, перемещение, удаление файлов и каталогов внутри файловой системы.

2.4 Архитектура разрабатываемого решения

Модель работы разрабатываемой библиотеки представляет собой клиент-серверное взаимодействие и представлена на рис. 1.

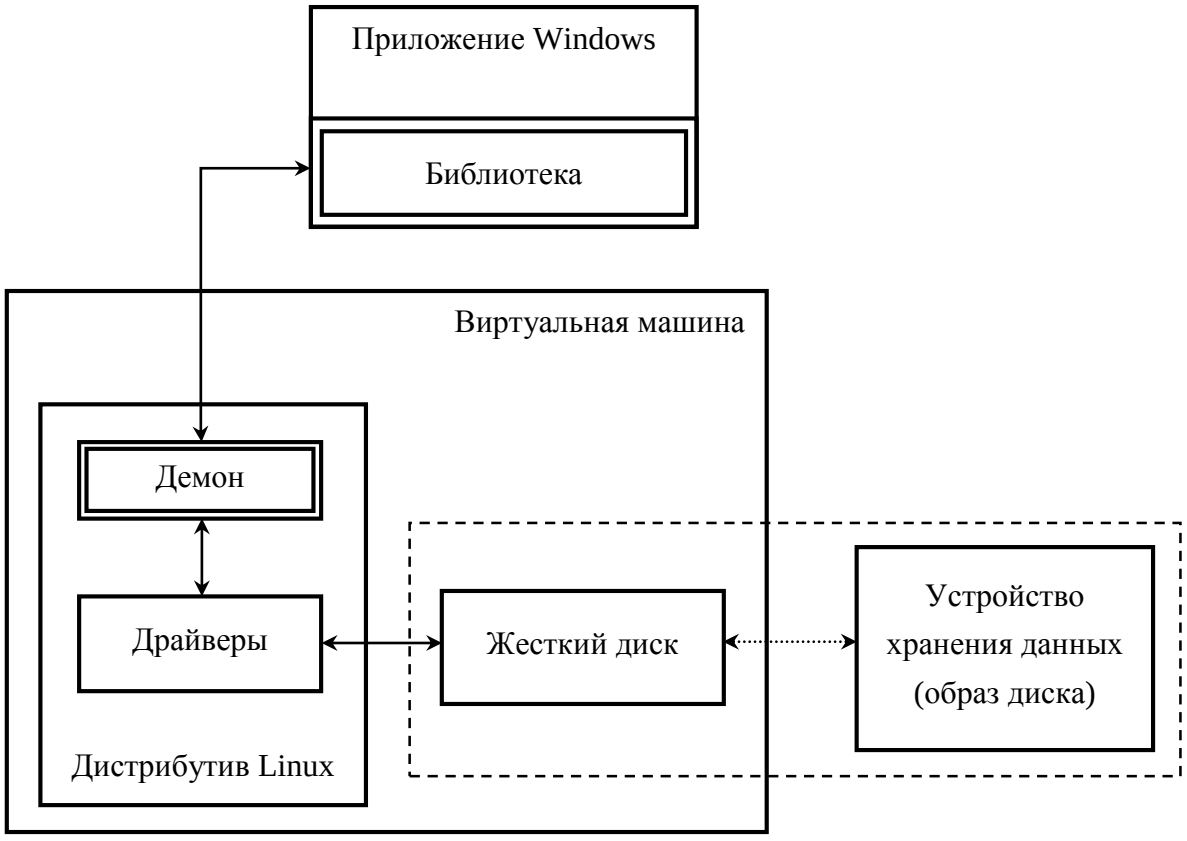

Рис. 1 Схема работы библиотеки

Библиотека, работающая на стороне приложения, в отдельном (дочернем) процессе запускает виртуальную машину, внутри которой загружается дистрибутив Linux. В состав загружаемого дистрибутива входят нативные драйвера файловых систем и определенный набор вспомогательных утилит для работы с ними. После загрузки операционной системы Linux в ней запускается специальный контролирующий демон, способный выполнять команды, полученные от библиотеки, по взаимодействию с требуемыми файловыми системами. Таким образом, библиотека, предоставляющая приложению определенный программный интерфейс по работе с файловой системой, выступает в роли клиента, а запущенная ею виртуальная машина с загруженной внутри операционной системой Linux и демоном выступает в роли сервера.

При запуске виртуальной машины к ней присоединяется физическое устройство (либо образ диска) с файловой системой, доступ к которой необходимо организовать. Внутри виртуальной машины присоединенное устройство выглядит как некоторое устройство хранения данных, например, как жесткий диск, с нативной для операционной системы Linux файловой системой. Стоит отметить, что в случае, если вместо реального физического устройства к виртуальной машине присоединяется образ диска, то виртуальная машины сама транслирует доступ ядра Linux к «физическому» устройству в доступ к образу диску.

Благодаря наличию в операционной системе Linux необходимых драйверов, работающий внутри виртуальной машины демон монтирует это устройство, после чего способен взаимодействовать с монтированной файловой системой и осуществлять требуемые операции чтения и записи в соответствии с командами, полученными от библиотеки, работающей на стороне приложения.

Библиотека, выступающая в роли клиента, «общается» с демоном с помощью механизма удаленного вызова процедур (RPC – Remote Procedure Call). Библиотека отправляет демону определенные команды для осуществления операций чтения и записи файловой системы и получает обратно от демона результаты их выполнения.

Рассмотренный механизм работы позволяет осуществить доступ к любой нативной файловой системе операционной системы Linux, поддержка которой включена в используемый в виртуальной машине дистрибутив Linux.

2.5 Проект libguestfs

Проект libguestfs – проект с открытым исходным кодом компании RedHat, активно развивается с 2009 года. Предназначен для использования в операционной системе Linux. Представляет собой набор утилит для доступа и модификации образов дисков виртуальных машин. Предоставляемые инструменты позволяют просматривать и редактировать файлы внутри гостевых систем, создавать образы для виртуальных машин, модифицировать их, ужимать, модифицировать таблицу разделов, управлять конфигурационными файлами,

переносить «железные» машины в виртуальную среду, переносить виртуальные машины с одного образа на другой, переносить виртуальные машины из образа на железо. Для выполнения своих операций libguestfs не требует прав суперпользователя, что является несомненным преимуществом.

Предоставляемый libguestfs набор инструментов позволяет получить доступ практически к любой из существующих файловых систем: поддерживаются все нативные файловые системы операционных систем Linux, Windows, Mac OS X, BSD, в том числе менеджер логических разделов (LVM2) Linux, дисковые разделы с таблицей разделов типа MBR и GPT, «сырые» (raw) образов дисков, CD и DVD диски, образы дисков формата ISO, SD-карты и многое другое. Также поддерживается работа с образами дисков, используемых современными виртуальными машинами. К ним относятся qcow2, VirtualBox VDI, VMWare VMDK, Hyper-V VHD/VHDX. Стоит отметить, что доступ к указанным файловым системам может осуществляться как локально, так и удаленно с помощью протоколов FTP, HTTP, SSH, iSCSI и некоторых других.

Особый интерес вызывает реализованная в проекте libguestfs библиотека, которая инкапсулирует в себе некоторую базовую функциональность, используемую во всех разработанных в рамках проекта инструментах. Данная библиотека предоставляет приложениям программный интерфейс для доступа и работы с упомянутыми ранее файловыми системами и образами дисков. Библиотека реализована на языке C, имеет привязки более чем к 10 языкам таким как Ocaml, Python, Ruby, Java, Haskell и лругие.

Модель и принцип работы библиотеки libguestfs

Предоставляемая проектом libguestfs библиотека в своей работе использует архитектуру, аналогичную рассмотренной в 2.4.

В качестве виртуальной машины используется QEMU[1] и UML[2]. Также есть возможность использовать libvirt либо подключиться к уже запущенному внутри виртуальной машины демону.

Взаимодействие библиотеки с демоном осуществляется через Unix-сокеты посредством удаленного вызова процедур по протоколу XDR. В большинстве случаев каждому вызову функции библиотеки соответствует отправка определенной команды демону.

Поскольку библиотека предназначена для использования в операционной системе Linux, в качестве запускаемого внутри виртуальной машины дистрибутива Linux используется дистрибутив, полученный с помощью утилиты supermin[3] из ядра операционной системы хоста и набора инструментов пространства пользователя (таких как программы LVM и Ext2) для работы с поддерживаемыми Linux файловыми системами. Также создается начальный RAM диск (initrd) для загрузки Linux и образ диска с файловой системой Ext2, используемый запущенной внутри виртуальной машины операционной системой Linux в качестве корневого.

[1] http://wiki.qemu.org/Main_Page
[2] http://user-mode-linux.sourceforge.net
[3] http://libguestfs.org/supermin.1.html

В документации к libguestfs вводится концепция «устройство» (appliance). «Устройство» представляет собой виртуальную машину с операционной системой Linux и запущенной внутри неё демоном и монтированным корневым образом диска.

Подготовка библиотеки к работе включает:

Процесс инициализации устройства аключает:

- Создание дистрибутива Linux для запуска внутри виртуальной машины
- Используется утилита supermin, разрабатываемая в рамках проекта libguestfs, для генерации ядра операционной системы Linux, начального RAM диска и корневого образа дика на основе файловой системы Ext2, содержащий набор необходимых драйверов, модулей и утилит.
- Запуск виртуальной машины
- Загрузка initrd
- Поиск и монтирование корневого образа диска, содержащего операционную систему Linux.
- Запуск демона

Для возможности внутри пользовательского приложения осуществлять настройку параметров запуска виртуальной машины и впоследствии взаимодействовать с запущенным внутри нее демоном библиотека libguestfs предоставляет специальный обработчик – объект типа “guestfs_h”, который создается при помощи вызова функции “guestfs_create”. Данный обработчик характеризует текущее состояние создаваемого библиотекой «устройства», которое представляет собой виртуальную машину с запущенным внутри неё демоном. Данный обработчик используется при вызове всех функций, предоставляемый библиотекой libguestfs в качестве программного интерфейса (API).

«Устройство» может находиться в одном из трех состояний:

- Конфигурация
- Запуск
- готовность к работе

Конечный автомат состояний «устройства» представлен на рис. 2.

/* схема конечного автомата состояний */

Правильная работа с «устройством» предполагает следующую очередность смены состояний: «конфигурация», «запуск», «готовность к работе».

Программный интерфейс библиотеки предлагает один способ перейти из состояния «конфигурация» через состояние «запуск» в состояние «готов к работе» - посредством вызова функции “guestfs_launch”. Вызов функции “guestfs_launch” блокирует выполнение приложения до тех пор, пока не закончится процесс инициализации устройства и оно не перейдет в состояние «готовность к работе». “guestfs_launch» в процессе своего вызова изменяет состояние устройства из «конфигурация» в «запуск», пока происходит запуск виртуальной машины, инициализация операционной системы и запуск демона.

После того, как «устройство» перешло в состояние «готовность к работе», можно осуществлять доступ к требуемой файловой системе. Для этого необходимо монтировать устройство, на котором она расположена, с помощью функции “guestfs_mount”, после чего

с помощью вызова соответствующих функций выполнять операции по чтению и записи файловой системы.

Задачи работы

Поскольку в рамках выбранного подхода организации доступа к нативным файловым системам операционной системы Linux в операционной системе Windows архитектура и принцип работы библиотеки libguestfs соответствует разрабатываемой, было принято решение портировать исходный код библиотеки libguestfs для возможности её работы в операционной системе Windows.

Таким образом, выполняемые в рамках настоящей работы задачи были сформулированы следующим образом:

1. Выбрать подходящую виртуальную машину для запуска операционной системы Linux внутри операционной системы Windows
2. Портировать библиотеку libguestfs для работы в операционной системе Windows
3. Предложить и реализовать ряд улучшений по повышению производительности и скорости работы библиотеки
4. Сравнить производительность работы портированной библиотеки с исходной

Выбор виртуальной машины

Основными кандидатами на использование в качестве виртуальной машин являлись VirtualBox, coLinux и QEMU. Рассмотрим особенности каждой из них более подробно.

VirtualBox

Программный продукт виртуализации для операционных систем Microsoft Windows, Linux, FreeBSD, Mac OS X, Solaris/OpenSolaris, ReactOS, DOS и других.

Ключевые возможности:

- кроссплатформенность
- поддерживает аппаратную виртуализацию
- высокая производительность работы гостевой операционной системы

coLinux

Технология, позволяющая нативно запускать операционную систему Linux на операционной системе Windows 2000/XP/Vista/7. coLinux использует модифицированный ядро Linux для запуска совместно с другой операционной системой на одной и той же машине. Такая возможность достигается благодаря использованию специального 32-битный драйвер Windows для отображения системных вызовов Linux в вызовы Windows.

Отличительные особенности:

- позволяет добиться практически той же функциональности и производительности как при работе обычной операционной системы Linux, запущенной на той же самой машине в одиночку
- требует прав суперпользователя, работает в привилегированном режиме процессора, в случае ошибок возможен крах всей системы
- не поддерживает работу в 64-битных операционных системах

QEMU

Свободно распространяемая виртуальная машина с открытым исходным кодом для эмуляции аппаратного обеспечения различных платформ. Включает в себя эмуляцию

процессоров Intel x86 и устройств ввода-вывода. Используется как чистый эмулятор и как нативная виртуальная машина (в x86 и x86-64 архитектурах). Работает в операционных системах Linux, Windows, Syllable, FreeBSD, FreeDOS, Mac OS X, QNX, Android и других.

В итоге, в качестве виртуальной машины было принято решение использовать QEMU, поскольку данная виртуальная машина обладает следующими преимуществами:

- свободно распространяется, имеется возможность сборки и использования в операционной системе Windows
- работает в пространстве пользователя, не требует прав администратора
- изначально поддерживается в библиотеке libguestfs

Портирование библиотеки libguestfs для работы в Windows

Выбор среды разработки

Портирование библиотеки началось с выбора подходящей среды разработки. Изначально хотелось использовать «родную» для операционной системы Windows среду разработки Visual C++, разрабатываемую компанией Microsoft и предназначенную для разработки нативных Windows-приложений на языке C++.

Однако использование Visual C++ оказалось невыгодным вследствие следующих причин:

- Отсутствие поддержки расширений языка C как в GCC

Поскольку проект libguestfs изначально разрабатывался для работы в Linux, сборка библиотеки libguestfs подразумевает использование инструментов GCC. Предлагаемые ими расширения языка C, такие как атрибуты функций, активно применяется в исходной коде библиотеки.

- Отсутствие поддержки системы сборки libguestfs

В проекте libguestfs применяется система сборки, основанная на использовании инструментов GNU Autotools. В случае использования для портирования библиотеки среды Visual C++ возникает необходимость в разработке собственной системы сборки при невозможности интегрироваться в уже существующую систему сборки проекта libguestfs.

Таким образом, отказ от использования Visual C++ предполагает необходимость использования таких инструментов разработки, которые лишены рассмотренных выше недостатков Visual C++, а именно предоставляют достойный Windows-аналог компилятора GCC и, желательно, поддерживают систему сборки GNU. С точки зрения выдвинутых требований наиболее подходящими средами разработки оказались Cygwin и MinGW.

Портирование libguestfs с использованием Cygwin

Cygwin – проект компании RedHat, представляет собой Unix-подобную среду и интерфейс командной строки для Microsoft Windows. Cygwin обеспечивает тесную интеграцию Windows приложений, данных и ресурсов с приложениями, данными и ресурсами UNIX-подобной среды. Ключевой частью Cygwin является его динамически подключаемая библиотека (DLL) cygwin1.dll, которая обеспечивает совместимость интерфейса прикладного программирования (API) и реализует значительную часть стандарта POSIX на основе системных вызовов Win32. Кроме того, Cygwin включает в себя инструменты разработки GNU для выполнения основных задач программирования, а также и некоторые прикладные программы, эквивалентные базовым программам UNIX.

Иными словами, Cygwin представляет собой удобную среду для портирования Unix-приложений в Windows практически без изменения их исходного кода. Это оказалось верным и для библиотеки libguestfs.

В рамках данного портирования были осуществлена сборка библиотеки libguestfs в Cygwin, которая включала следующие этапы:

- Изучение системы сборки проекта libguestfs
- Внесение изменений в систему сборки проекта libguestfs

  Поскольку главной целью является сборка библиотеки libguestfs, генерация и сборка многих утилит, реализованных в рамках данного проекта, была отключена. Также была отключена проверка некоторых зависимостей, которые не являются критичными для сборки библиотеки.

- Разрешение внешних зависимостей

  Проект libguestfs имеет большое количество зависимостей от сторонних библиотек. Многие из них были разрешены с помощью встроенного в установщик Cygwin менеджера пакетов. В случае отсутствия библиотеки (либо утилиты) в менеджере пакетов поиск и сборка библиотеки осуществлялась «вручную». Среди таких зависимостей оказались утилита supermin, менеджер пакетов ocamlfind.

В исходный код библиотеки libguestfs потребовалось внести минимальное количество изменений. Была изменена процедура запуска виртуальной машины: в связи с отсутствием в используемом Windows-дистрибутиве QEMU поддержки Unix-сокетов, они были заменены на TCP-сокеты. Также был отключен вызов утилиты supermin, который в исходной реализации библиотеки происходил с целью сгенерировать дистрибутив Linux для его использования в запускаемой виртуальной машине. Вместо этого поиск заранее подготовленного дистрибутива Linux происходил по пути, указанному в директиве GUESTFS_DEFAULT_PATH.

В результате данного портирования была реализована возможность использования библиотеки libguestfs приложениями операционной системы Windows, работающими в среде Cygwin.

Наряду с полученным положительным результатом, существенным недостатком данного портирования оказался тот факт, что после установки всех необходимых библиотек и утилит размер дистрибутива Cygwin оказался слишком большим – порядка 1 ГБ. С целью устранить зависимость портированной библиотеки от Cygwin-окружения было принято решение осуществить нативное портирование библиотеки libguestfs с использованием инструментария, предлагаемого в рамках проекта MinGW.

Нативное портирование библиотеки libguestfs с помощью MinGW

Локализация платформозависимого кода и сокрытие его внутри простых кроссплатформенных интерфейсов, оформленных в виде отдельной библиотеки.

Изменена процедура получения дистрибутива Linux

В исходной реализации библиотеки libguestfs перед запуском виртуальной машины происходит вызов утилиты supermin, в результате которого создается дистрибутив Linux, используемый впоследствии внутри виртуальной машины. Дистрибутив Linux генерируется на основе Linux, установленного на хосте. Поскольку случае операционной

системы Windows вызов утилиты supermin является бессмысленным и, очевидно, приведет к ошибке, были изменена процедура получения дистрибутива Linux, а именно:

Отключен вызов утилиты supermin

Вместо генерируемого утилитой supermin дистрибутива Linux используется заранее подготовленный дистрибутив Linux, расположенный по пути, указанному в “GUESTFS_DEFAULT_PATH/appliance”.

изменено на поиск в заранее указанном месте уже построенного дистрибутива.

Портирование модуля command

Command

Для исполнения системных команд операционной системы в библиотеке используется абстракция command. Объект типа command используется для выполнения системных вызовов и получения результатов их исполнения. Объект типа command позволяет также получить данные стандартного потока вывода и стандартного потока вывода ошибок.

В реализации command для запуска отдельного процесса используется вызов “fork”, внутри которого происходит вызов “execv” с наследуемой от процесса-родителя командой в виде строки. Данной семантике вызова “fork” хорошо подходит предлагаемая API Windows функция CreateProcess.

Для получения данных потока стандартного вывода и потока стандартного вывода ошибок использовались анонимные каналы, позволяющие назначить в качестве стандартного вывода один из концов анонимного канала.

В библиотеке libguestfs для выполнения системных команд используется абстракция command. Она представляет собой некоторую обертку над функциями “execv” и “system” и используется для получения данных с потоков стандартного вывода и стандартного вывода ошибок, генерируемых во время выполнения внешней команды. В исходной реализации библиотеки libguestfs для вызова внешней команды используется fork с последующим вызовом “execvp” (либо “system”). Для получения данных со стандартных потоков используются однонаправленные POSIX каналы, создаваемые с помощью вызова функции pipe.

Используемый для вызова внешней команды fork с последующим вызовом функции “execv” (либо “system”) был заменен на CreateProcess. Для получения данных со стандартных потоков исполнения использовался предлагаемый Windows механизм анонимных каналов с возможностью их наследования дочерним процессом, создаваемым путем вызова CreateProcess, в качестве потоков стандартного вывода и стандартного вывода ошибок.

? /* код с CreateProcess и перенаправлением потоков вывода */ ?

Портирование модуля conn-socket

Сетевое взаимодействие, используемое для общения с демоном, инкапсулируется абстракцией connection_socket

Для описания сетевого соединения библиотеки с демоном, работающего внутри виртуальной машины, используется структура struct connection_socket. Поле daemon_accept_sock описывает прослушивающий сокет для установления соединения с демоном и daemon_sock используется для взаимодействия с демоном после установки

соединения. Поле console_sock предназначен для получения данных с потоков стандартного ввода, стандартного вывода и стандартного вывода ошибок и представляет собой один их концов двунаправленного канала, создаваемого с помощью socketpair.

Исходная и полученная кроссплатформенная реализации приведены ниже.

В случае операционной системы Windows поля daemon_sock и daemon_accept_sock представлены типом SOCKET. Поле console_sock описывается типом HANDLE и представляет собой один из концов анонимного канала. В соответствии с приведенными изменениями была создана отдельная реализация всех функций модуля conn-socket.c для работы в Windows.

Изменение процедуры подготовки и запуска виртуальной машины

Подготовка необходимого окружения и запуск виртуальной машины описываются, главным образом, функцией launch_direct, расположенной внутри модуля launch-direct.c. В работе данной функции можно выделить следующие этапы:

- Создание прослушивающего сокета
- Создание канала для получения данных с потоков стандартного ввода, вывода и вывода ошибок дочернего процесса, внутри которого запускается виртуальная машина
- Формирование и настройка параметров запуска виртуальной машины
- Создание дочернего процесса и запуск внутри него виртуальной машины с предварительным перенаправлением стандартных потоков (ввода, вывода, вывода ошибок) в один из концов канала, созданного на предыдущем этапе
- Принятие соединения от демона на прослушивающем сокете и создание сокета для взаимодействия с демоном

Поскольку внутри каждого из этапов используются платформозависимые возможности операционной системы, которые отличаются в операционных системах Windows и Linux, реализация каждого из этапов была скрыты внутри кроссплатформенной функции и портирована для работы в операционной системе Windows (в случае операционной системы Linux исполняется исходный код библиотеки libguestfs). Особенности реализации Windows-версии каждого из этапов приведены ниже.

Реализация отдельных POSIX функций и расширений GNU для работы Windows

Используемые в реализации библиотеки функции, предоставляемые в рамках стандарта POSIX или относящиеся к расширениям GNU, аналогов которым нет в операционной системе Windows, были реализованы на основе приведенной к ним в сети Интернет документации и оформлены в виде отдельной библиотеки операционной системы Windows.

В состав полученной библиотеки вошли следующие функции:

asprintf, vasprintf, symlink, strndup, realpath, memmem, open_memstream, getusername, getusid, getuid, geteuid, strerror_r

Реализация кроссплатформенных интерфейсов

В ходе портирования была создана библиотека, содержащая в себе кроссплатформенные определения таких базовых концепций операционных систем как процесс (os_process_info_t), сокет (os_socket_t, os_socket_info), однонаправленный канал (os_pipe_t, os_pipe_end_t), двунаправленный канал (os_socketpair_t, os_socketpair_end_t). Для каждой из приведенных структур были реализованы вспомогательные функции, реализующие инициализацию, проверка на валидность, присвоение значения по умолчанию и некоторые уникальные для структуры операции (os_process_info__kill, os_pipe_end__close и т.д.).

Интеграция в систему сборки проекта libguestfs

Как упоминалось ранее, проект libguestfs использует систему сборки GNU Autotools. Данная система сборки основана на запуске специального скрипта configure, результатом исполнения которого является файл конфигурационный config.h и получаемое на основе файлов Makifile.in дерево Makefile-ов. Более подробно о принципах работы данной системы сборки рассказывается в [?].

В исходный процесс сборки был добавлен режим “mingwport”, для включения которого в используется опция “--enable-mingwport”, передаваемая в качестве одного из входных параметров скрипта configure. В режиме “mingwport” отключена проверка зависимостей, не являющимися необходимыми в рамках сборки библиотеки, а также отключена сборка многих инструментов, реализуемых в рамках проекта. В данном режиме происходит только сборка основной библиотеки libguestfs и добавленных в проект вспомогательных библиотек winport и osdep. В операционной системе Linux в режиме “mingwport” также происходит сборка демона и построение используемого в виртуальной машине дистрибутива Linux с помощью вызова утилиты supermin.

Поскольку библиотека libguestfs изначально предоставляет гораздо больший функционал, чем требуется в рамках решения обозначенной в данной работе проблеме, главной целью портирования являлось получение библиотеки, использующей для своей работы одну из наиболее «подходящих» виртуальных машин [?] и обеспечивающей приложениям операционной системы Windows возможность монтирования нативных файловых систем операционной системы Linux, выполнения операций чтения и записи файловых систем.

Портирование библиотеки libguestfs осуществлялось в несколько этапов:

- Портирование библиотеки libguestfs для запуска в Cygwin
- Нативное портирование библиотеки libguestfs на Windows
- Нативное портирование библиотеки libguestfs при помощи MinGW

Cygwin

На внутреннем форуме компании RedHat уже поднимался вопрос о необходимости портировать библиотеку libguestfs для работы в Windows. В ходе обсуждения разработчиками была высказана идея о целесообразности начать портирование именно с реализации возможности запуска в Cygwin, поскольку это потребует меньшее количество усилий по сравнению с нативным портированием, а также позволит оценить сложность и возможность портирования библиотеки без необходимости полного её переписывания.

Выбор виртуальной машины

Результаты портирования

Портирование на Cygwin не потребовало внесения существенных изменений в исходный код библиотеки. Практически все Linux-зависимые функции, используемые в реализации libguestfs, поддерживаются в Cygwin.

В ходе портирования были произведены:

Изменения в системе сборки проекта libguestfs

В проекте libguestfs используется система сборки GNU Autotools [?]. Поскольку целями портирования является поддержка возможности монтирования файловой системы, чтения и записи файлов, генерация и сборка многих утилит была отключена. Также была отключена проверка некоторых зависимостей, которые не являются критичными для сборки библиотеки.

Разрешение зависимостей проекта

Проект libguestfs имеет большое количество зависимостей от сторонних библиотек. Многие из этих библиотек доступны через поставляемый вместе с Cygwin менеджер пакетов. В случае, если библиотека недоступна через менеджер пакетов, тогда требуется вручную собирать её из исходников. Поскольку библиотеки, используемые в libguestfs, предназначены, главным образом, для компиляции и использовании в Linux, процесс их сборки в операционной системе Windows не всегда оказывается тривиальным.

Изменения параметров запуска виртуальной машины

В качестве виртуальной машины использовался исполняемый дистрибутив QEMU, предназначенный для запуска в Windows. Данный дистрибутив был получен с сайта [?]. Обоснование выбора виртуальной машины приведено в [?].

Как упоминалось ранее, в библиотеке libguestfs для взаимодействия с демоном используется сетевое взаимодействие. Для его реализации в libgustfs используется unix-сокет, путь к которому указывается в качестве одного из параметров виртуальной машины непосредственно перед её запуском. Поскольку Windows не поддерживает концепцию unix-сокетов, была создана кроссплатформенная реализации настройки параметров запуска виртуальной машины QEMU, которая в случае операционной системы Windows создает и использует для сетевого взаимодействия tcp-сокет.

Тестирование производительности

Нативное портирование библиотеки libguestfs на Windows

Нативное портирование библиотеки libguestfs на Windows при помощи MinGW

Портирование включало в себя как реализацию Windows-аналогов отдельных функций, так и переписывание отдельных модулей.

В ходе портирования функции, специфичные для операционной системы Linux и реализованные для запуска в Windows, были реализованы в виде отдельной Windows-библиотеки.

В состав полученной библиотеки были включены Windows-аналоги следующих функций:

asprintf, vasprintf, symlink, strndup, realpath, memmem, open_memstream, getusername, getusid, getuid, geteuid, strerror_r. Данные функции были реализованы на основе приведенной в сети Интернет документации к ним.

conn-socket

Сетевое взаимодействие, используемое для общения с демоном, инкапсулируется абстракцией connection_socket. Сокеты используемые внутри неё были заменены кроссплатформенной реализацией.

XDR

Для реализации удаленного вызова процедур используется протокол XDR. Для возможности его использования в Windows использовалась библиотека, реализованная в рамках проекта bsd-xdr [?].

Protobuf

В рамках портирования была предпринята попытка заменить протокол удаленного вызова процедур. Причиной послужило отсутствие официальной реализации XDR от Windows. Возникло желание заменить XDR современным протоколом, находящийся в непрерывном развитии и активно поддерживающийся разработчиками. В качестве альтернативной замены протокола XDR был выбран протокол Protobuf от компании Google.

Protocol Buffers – гибкий эффективный механизм для сериализации структур данных – наподобие XML, но меньше, быстрее и проще. Данный протокол предназначен для использования в C++, Java и Python.

Существует реализация Protobuf для языка C – проект c-protobuf на github [?]. Данный проект не является официально разрабатываемым проектом от Google, но в его разработке принимают участие никто. Данный проект перечислен как один из рекомендуемых на сайте google [?] https://github.com/google/protobuf/wiki/Third-Party-Add-ons

К сожалению, замена XDR на Protobuf не принесла выигрыша в производительности. Более того, тестирование показало ухудшение в скорости работы.

Реализация кроссплатформенной библиотеки

Appliance – remove build

Перед запуском виртуальной машины библиотека libguestfs запускает supermin, который строит загружаемый в виртуальной машине дистрибутив Linux. Поскольку в случае работы в операционной системе Windows такое невозможно, данное поведение было изменено на поиск в заранее указанном месте уже построенного дистрибутива.

Error – perrorf_wsa, perrorf_win

Launch-direct.c

Создана кроссплатформенная реализация метода запуска виртуальной машины

Launch – get_umask

Wsa_init

Osdep

Uio

Win-port

Выбор виртуальной машины

Улучшение производительности работы библиотеки

С целью повысить скорость выполнения операций чтения и записи файловой системы были реализованы следующие подходы:

- Замена протокола взаимодействия с XDR на ProtoBuf
- Передача файлов через общую память

Замена протокола взаимодействия с XDR на ProtoBuf

Передача файлов через общую память

«Общение» библиотеки с демоном осуществляется посредством сетевого взаимодействия, все данные – команды и результаты их выполнения – передаются через сокеты. В частности, файлы, отправленные демону на запись или полученные в результате выполнения операции чтения, пересылаются частями – максимум по 8 КБ в каждом пакете. Учитывая, что пересылка каждого пакета сопровождается дополнительными издержками, связанными с организацией и структурой сетевого взаимодействия (копирование данных для упаковки в пакет, передача пакета, распаковка пакета и извлечение данных), это приводит к существенному замедлению скорости выполнения операций чтения и записи, особенно при передаче файлов большого (порядка нескольких десятков мегабайт и выше) размера. С целью повысить скорость работы библиотеки было принято решение организовать передачу файлов через разделяемую (общую) память.

Использование разделяемой памяти позволит:

Передавать содержимое файлов частями гораздо большего размера без увеличения дополнительных издержек, связанных с передаваемым размером данных

Избавиться от «лишнего» копирования, происходящих при передаче данных через сокеты

Организация доступа библиотеки к разделяемой памяти

Организация доступа к разделяемой памяти со стороны библиотеки достаточно проста. В операционной системе Windows разделяемая память между процессами организуется с помощью файловых отображений. Файловые отображения относятся к глобальным объектам операционной системы Windows и обладают собственным уникальным именем. Для получения объекта файлового отображения в библиотеке используется вызов функции OpenFileMapping, в котором указывается имя файлового отображения. Далее с помощью вызова функции MapViewOfFile участок общей памяти отображается в используемое библиотекой виртуальное адресное пространство, а полученный в результате вызова указатель используется в качестве буфера для передачи и получения файлов.

Для работы с разделяемой памятью (на стороне библиотеки) была реализована кроссплатформенная структура os_shared_memory, которая была добавлена в созданную библиотеку osdep кроссплатформенных интерфейсов. Данная структура позволяет открывать разделяемую память, получать указатель на начало разделяемой памяти, а также хранит сведения о свойствах разделяемой памяти, таких как размер и имя.

/* листинг структуры */

В случае операционной системы Windows в качестве структуры os_shared_memory выступает структура windows_shared_memory, которая предоставляет интерфейс по работе с файловыми отображениями операционной системы Windows.

Таким образом, в исходный код библиотеки были внесены следующие изменения:

В открытый программный интерфейс библиотеки добавлена функция guestfs_set_shared_memory, предоставляющая возможность включить (отключить) использование общей памяти для передачи файлов, а также задать имя и размер

Поскольку состояние библиотеки теперь характеризуется использованием общей памяти, в «главный» обработчик “guestfs_h” было добавлено поле shmem типа os_shared_memory

В процедуру запуска виртуальной машины добавлена проверка необходимости использования разделяемой памяти и при наличии таковой во входные параметры виртуальной машины включается соответствующая опция ivshmem

Изменен протокол передачи данных для случая использования общей памяти

https://msdn.microsoft.com/en-us/library/windows/desktop/aa366878(v=vs.85).aspx

Используя описанную выше процедуру, библиотеке, работающей внутри использующего её приложения операционной системы windows, достаточно просто получить указатель на разделяемую память. Наибольший интерес вызывает организация получения доступа к разделяемой памяти Windows демоном, работающего внутри виртуальной машины с операционной системой Linux.

Организация доступа демона к разделяемой памяти

Организация доступа к разделяемой памяти со стороны демона выглядит немного сложнее, чем в случае с библиотекой, работающей на стороне Windows-приложения. Демон работает в гостевой операционной системе Linux внутри виртуальной машины, не имеет доступа к операционной системе Windows, в которой запущена виртуальная машина, и, как следствие, не может напрямую использовать файловые отображения операционной системы Windows. В данном случае необходимо, чтобы запущенная в Windows виртуальная машина QEMU, обладая доступом к разделяемой памяти Windows, представила эту общую память в виде некоторого устройства, доступного демону в операционной системе Linux внутри виртуальной машины.

Устройство IVSHMEM виртуальной машины QEMU

В документации к виртуальной машине QEMU указывается о возможности использовать общую память между хостом и гостем. Внутри виртуальной машины общая память представляется гостевой операционной системе в виде памяти особого PCI-устройства, позволяющего приложениям взаимодействовать с ним «без копирования» (zero-copy communication). Данное устройство носит название ivshmem (Inter-VM Shared Memory).

Однако данная возможность доступна только при использовании виртуальной машины QEMU в операционной системе Linux. Поэтому в рамках решения задачи организации доступа демона к разделяемой памяти необходимо реализовать эту возможность при использовании виртуальной машины QEMU в операционной системе Windows.

Реализация IVSHMEM-устройства для работы в Windows

В исходном коде проекта QEMU реализация ivshmem-устройства содержится в модуле ivshmem.c. Для описания ivshmem-устройства и его состояния используется структура IVShmemState, наследуемая от структуры PCIDevice. Установка разделяемой памяти хоста в качестве памяти ivshmem-устройства происходит во время инициализации устройства. Таким образом, для реализации возможности использования ivshmem-устройства в Windows были внесены изменения в процедуру его инициализации.

Во время инициализации ivshmem-устройства в функции pci_ivshmem_init происходит создание файлового отображения (разделяемой памяти) с помощью функции CreateFileMapping. После создания файлового отображения внутри вызова функции create_shared_memory_BAR с помощью функции MapViewOfFile возвращается указатель на отображенную в виртуальное адресное пространство QEMU общую память, которая регистрируется в качестве физической памяти созданного PCI-устройства. Имя

создаваемого файлового отображения, указываемое в функции CreateFileMapping, передается в качестве входного параметра при запуске виртуальной машины QEMU. Листинг pci_ivshmem_init

Листинг create_shared_memory_BAR

Для реализации ivshmem-устройства в Windows использовался исходный код QEMU версии 2.1.2, полученный с официального сайта [?].

http://wiki.qemu.org/Download

Сборка QEMU осуществлялась с использованием MinGW в соответствии с указаниями, представленными на сайтах [?].

http://wiki.qemu.org/Hosts/W32

http://lassauge.free.fr/qemu/QEMU_on_Windows.html

Организация доступа демона к ivshmem-устройству

Для возможности работы в операционной системе Linux с ivshmem-устройством, представленным как PCI-устройство в виртуальной машине, в используемый дистрибутив Linux был добавлен специальный драйвер, который загружается в ядро во время выполнения начального загрузочного скрипта. Данный драйвер реализован в рамках проекта guest_code [?] с использованием фреймворка UIO ядра Linux. Также в начальный загрузочный

Таким образом, в исходный код демона были внесены следующие изменения:

Реализована структура shared_memory, предназначенная для работы с памятью ivshmem-устройства

В процесс инициализации демона (подготовки демона к работе) добавлена проверка необходимости использования общей памяти, и при наличии таковой осуществляется поиск ivshmem-устройства и отображение его памяти в виртуальное адресное пространство демона (метод open структуры shared_memory)

Изменен протокол передачи (получения) файлов [?] для случая использования общей памяти

https://gitorious.org/nahanni/guest-code/

Изменения протокола передачи файлов

В исходной реализации файлы между библиотекой и демоном передавались через сокеты по частям, максимум по 8 КБ в каждом пакете. При таком способе передачи отсутствует необходимость ожидать, пока получателем будет считана в файл переданная часть, поскольку каждая следующая часть передается в отдельном пакете и не перезаписывает предыдущую.

В случае использования общей памяти для передачи файлов это не так. «Передача» очередной части файла осуществляется путем её записи в общую память. Для удобства рассмотрим процедуру передачи файла от библиотеку к демону. Отправитель (в данном случае это библиотека) осуществляет запись передаваемой части файла в общую память, после чего отправляет получателю (в данном случае - демону) сообщение-уведомление (guestfs_shm_chunk) о том, что переданная часть файла готова к получению. После получения уведомления получатель считывает содержимое общей памяти в создаваемый (обновляемый) файл и отправляет сообщение-подтверждение об успешном получении переданных данных. Только после получения подтверждения отправитель передает

следующую часть файла, поскольку если он «отправит» следующую часть файла до момента считывания получателем предыдущей, то новая часть файла перепишет содержимое предыдущей, в результате содержимое полученного файла окажется некорректным.

Результаты тестирования производительности

Главной задачей проведенного тестирование являлась сравнение скорости выполнения операций чтения/записи файлов большого и малого размера при использовании общей памяти и без неё. Тестирование

Тестирование производительности включало в себя:

Чтение 1 файла размером 1.5 ГБ

Чтение 10000 файлов размером 4 КБ каждый

Запись 1 файла размером 1.5 ГБ

Запись 10000 файлов размером 4 КБ каждый

Тестирование проводилось как в операционной системе Windows, так и в операционной системе Linux. Размер используемой общей памяти – 256 МБ. Результаты тестирования приведены в табл. [?], данные указаны в секундах.

	Linux	Linux: общая память	Windows	Windows: общая память
Чтение 1 ф. х 1.5 ГБ	46.1 сек.	13.3 сек.	47.8 сек.	13.0 сек.
Чтение 10000 ф. х 4 КБ	16.4 сек.	16.1 сек.	17.9 сек.	17.3 сек.
Запись 1 ф. х 1.5 ГБ	51.0 сек.	25.7 сек.	51.0 сек.	27.3 сек.
Запись 10000 ф. х 4 КБ	24.3 сек.	24.2 сек.	26.4 сек.	27.2 сек.

Как видно из полученных результатов, в случае передачи файлов малого размера (до 8 КБ), содержимое которых помещается в один пакет в случае передачи без использования общей памяти, скорость выполнения операций чтения/записи существенно не изменяется. В случае передачи файлов большого размера (порядка нескольких десятков мегабайт и выше), мы наблюдаем огромный прирост в производительности: скорость записи возрастает почти в 2 раза, скорость чтения – более, чем в 3 раза.

Стоит отметить, что в случае использования общей памяти скорость передачи файлов также существенно зависит от размера общей памяти – чем больше размер используемой разделяемой памяти, тем выше скорость передачи файлов больше. Причем прирост в скорости передачи файлов от увеличения размера используемой общей памяти будет происходить только тогда, пока размер общей памяти не превосходит размер передаваемого файла.

Результаты работы

В настоящей работе получена библиотека, позволяющая приложениям операционной системы Windows работать с нативными файловыми системами операционной системы Linux, которые не поддерживаются операционной системой Windows. Полученная библиотека представляет собой портированную в Windows версию библиотеки libguetfs, реализованную в рамках одноименного проекта.

Организация доступа к файловым системам основана на использовании нативных драйверов операционной системы Linux, которые включены в её основное ядро. Для возможности работы этих драйверов в операционной системе Windows окружение Linux эмулируется с помощью виртуальной машины. В качестве виртуальной машины используется QEMU.

Для повышения скорости выполнения операций чтения и записи реализована возможность передачи файлов через общую память.

Произведено сравнение производительности работы портированной и исходной библиотеки libguestfs

Произведен

Главной задачей тестирования была проверка работоспособности библиотеки. Также оценивалось изменение производительности работы библиотеки по сравнению с её работой в операционной системе Linux.

Производительность работы библиотеки оценивалась по следующим показателям

- Время от момента запуска виртуальной машины до полной инициализации утройства
- Время монтирования файловой системы
- Время выполнения операций чтения

Таким образом, тестирование производилось как в операционной системе Linux, так и в среде Cygwin, и включало в себя:

- Запуск виртуальной машины с 256 МБ памяти
- Монтирование образа диска с файловой системой Ext4
- Чтение 1 файла размером 1.5 ГБ из монтированной файловой системы
- Чтение 1000 файлов размером 4 КБ с монтированной файловой системы
- Размонтирование диска, выключение виртуальной машины

Результат тестирования приведены в виде диаграммы на рисунке [?].

/* графики, на которых Cygwin намного хуже по скорости чтения файлов: мелких и больших. По скорости записи файлов: мелких и больших. По потреблению оперативной памяти: потребление главным образом за счет запуска qemu */

Как видно из полученных результатов, производительность работы библиотеки снизилась. В Cygwin скорость выполнения всех тестируемых операций уменьшилась. Особенно увеличилось время инициализации «устройства» и время чтения файла большого размера.

Полученные результаты достаточно предсказуемы, поскольку:

- наличие «Cygwin-прослойки» предполагает дополнительные издержки
- в Linux QEMU использует KVM (в Windows нет поддержки KVM)
- библиотека libguestfs использует специфичные для Linux функции, портированные аналоги которых могут работать намного медленнее (например, функция fork)

Полученные показатели потребления оперативной памяти системы играют вторичную роль, поскольку сильно зависят от параметров запуска виртуальной машины, в частности, от того, сколько физической памяти будет предоставлено гостевой операционной системе

внутри виртуальной машины. В данном случае можно утверждать, что потребление оперативной памяти существенно не различается.

Полученный результат носит в большей степени экспериментальный характер и служит доказательством возможности портировать базовую часть библиотеки libguestfs, отвечающей за доступ к файловым системам, для работы в Windows.

Поскольку в случае успешной отправки и получения пакета

причем передача каждой следующей части происходила без подтверждения того, что предыдущая часть успешно принята. Благодаря наличию сокетного буфера, чтение пакетов на стороне получателя могло происходить медленнее их отправки, то есть реализация позволяла отправить несколько частей файла подряд до момента чтения первой части из этой партии.

Использование разделяемой памяти предполагает иной подход. Поскольку для передачи частей файла используется один «буфер» (в нашем случае это участок разделяемой памяти), отправка последующей части возможна только после того, как предыдущая часть была полностью считана из буфера – разделяемой памяти (в противном случае последующая часть перезапишет предыдущую и в итоге мы не получим исходный файл).

Реализация возможности использования разделяемой памяти требует внесения изменений как в исходный код библиотеки, используемой на стороне приложения, так и в исходный код демона.

Изменения в библиотеки

Изменения в демоне

Добавлены соответствующие изменения в процесс загрузки операционной системы и демона внутри виртуальной машины. Перед запуском демона происходит загрузка драйвера, отвечающего за взаимодействие с ivshmem-pci-устройстовом.

Далее в демоне происходит открытие устройства и отображение его памяти в виртуальное адресное пространство демона. Поскольку память pci-устройства и является разделяемой памятью, таким образом, демон получает доступ к разделяемой памяти.

За получение и отправку файлов в работе демона используются функции receive_file и send_file_write. Реализация данных файлов теперь позволяет работать в 2-ч режимах – с использованием разделяемой памяти и без неё.

Архитектура разрабатываемой библиотеки

==========
==========
==========

TODO: build_appliance incorrect appliance_path, demon uio get not hardcoded uio0

Текущая реализация qemu не предусматривает возможность использования ivshmem устройств при сборке в оп

В оригинальной библиотеке libguestfs все команды и результаты их выполнения передаются посредством сетевого взаимодействия. В случае выполнения операции чтения либо записи файла это существенно замедляют скорость работы.

Данные передаются пакетами. В случае чтения либо записи файла он передается частями «кусками» (chunk), размер одного куска составляет 8 КБ. Поскольку передача каждого пакета сопровождается накладными расходами связанными с упаковкой пакета, его пересылкой и распаковкой, скорость выполнения операций чтения и записи очень далека от нативной.

Вследствие этого было принято решение использовать общую память между гостем и хостом, а по сети передавать уведомления о готовности к получению новой части файла.

К вновь появляющимся файловым системам предъявляются

Использовать нативный драйвер файловой системы для той операционной системы, в которой эта файловая система является нативной.

Как видно из примеров, среди всех решений проблемы доступа к файловым системам ОС Linux внутри ОС Windows можно выделить два основных подхода: независимое приложение ОС Windows либо полноценный драйвер, работающий в режиме ядра, обеспечивающий нативный доступ к файловой системе.

Либо полноценные драйвера, работающие в режиме ядра операционной системы Windows, предоставляющие возможность осуществлять чтение и запись на диск с файловой системой из-под любой программы, но надежность некоторых из которых зачастую оставляет желать лучшего. Отдельные графические приложения и утилиты командной строки как правило предоставляют доступ к файловой системе только в режиме чтения для просмотра и копирования её содержимого. При этом отсутствует возможность доступа к данным из других программ. При использовании данного подхода для работы с файлами внутри файловой системы из других программ необходимо заранее вручную скопировать необходимые файлы на раздел диска с нативной для Windows файловой системой, после чего файл становится виден внутри любой программы, использующей путь, начинающийся с символа данного раздела (drive letter).

Use BTRFS:

coLinux

zfs-win : read-only https://code.google.com/p/zfs-win/

win-btrfs

virtual machine + network server

Файловые системы: значение, функции, виды

Windows ФС. Устройство, виды

Linux ФС. Многообразие, сравнение с Windows, необходимость поддержки. Наличие возможности Linux -> Windows, но отсутствие Windows -> Linux.

НЕОБХОДИМОСТЬ использования ФС Linux из-под Windows. Типы ПО, требующие (либо реализующие для своих нужд) данную функциональность.

Существующие решения. Основные подходы. Self-contained приложения, драйвера, read-only, виртуальные машины coLinux. Пример с хабра, возгласы пользователей.

Выбор подхода с виртуальной машиной. Обоснование. Libguestfs.

Необходимость поддержки ненативных файловых систем